by Philip Nowe

Low-Cost Logic Analyzer for FPGAs

Imagine being able to look inside your FPGA to see what it's doing. Now imagine doing so at a reasonable cost! You can do it all with Philip's Excel logic analyzer (ELA).

Last December, I described my Spartan-3 experimenter's board (SEB3) to show you how easy it is to design with an FPGA ("An FPGA Experimenter's Board," *Circuit Cellar*, 173, 2004). One of the great things about Xilinx's small Spartan-3 FPGA is that its tools are free. But what happens when you need to debug your design? How do you debug a design embedded in the FPGA?

There are a few different methods for debugging an FPGA design. The most important involves simulating your design with a simulator like Mentor Graphics's ModelSim, a free version of which is included in Xilinx's WebPACK. A simulator can give you the best insight into a digital design, but it isn't always easy to build an effective test bench.

Another issue is that many designers of board-level products aren't comfortable with digital simulation. Their method of getting a board up and running is to assemble the board (or evaluation boards), write the code, instrument the design (with real instruments), and wait to see what happens. This is also likely true of hobbyists, most of whom don't always have DSOs and logic analyzers. But the insight that such instruments offer is phenomenal. A logic analyzer could be your best friend, but it would be an expensive one. Even the low-cost USB/PC versions are pricey and can't look into your FPGA design. The signals of interest would have to be routed to external pins and then connected to the instrument. This may be possible, but you'll often want to look at more signals than you have spare pins for.

Placing a logic analyzer inside the FPGA is a reasonable alternative. Xilinx's ChipScope Pro embedded logic analyzer works extremely well, but it costs nearly $700. What to do? Try my Excel logic analyzer (ELA).

EMBEDDED ANALYZER

Some of the features that I was trying to include in my simple logic analyzer were 16 input channels, a 1,024-sample depth, a 100-MHz sample rate, a match unit with a match input and a mask input, and a fixed number of samples stored before and after the trigger. I know the embedded logic analyzer also would have to communicate to a PC where the results would be displayed, so I decided to use a serial port. Although it isn't particularly fast, it's available on many FPGA board designs. I developed the graphical waveform display with Visual Basic for Applications (VBA) within Microsoft Excel 2000.

Now I'll cover the ELA's design. First, I'll describe the FPGA circuitry and explain how to use Excel as the logic analyzer's user interface. Then I'll describe the test circuitry and explain how you can use the logic analyzer in your own design.

ELA DESIGN

The ELA has two hardware parts. The logic capture unit is the circuitry that samples the input channels and saves the results in RAM. It also determines when a match has taken place and when to stop data acquisition. The second part is a controller that communicates with the PC and logic capture unit. If you read my December article, it shouldn't surprise you that I used a Xilinx PicoBlaze for the controller. Figure 1 shows the block diagram of the PicoBlaze and the logic capture unit.

I started the logic capture unit's design with the match unit, which is composed of 16 single-bit compare modules. Each single-bit compare module looks at one of the input signal bits and compares it to the corresponding bit in the match

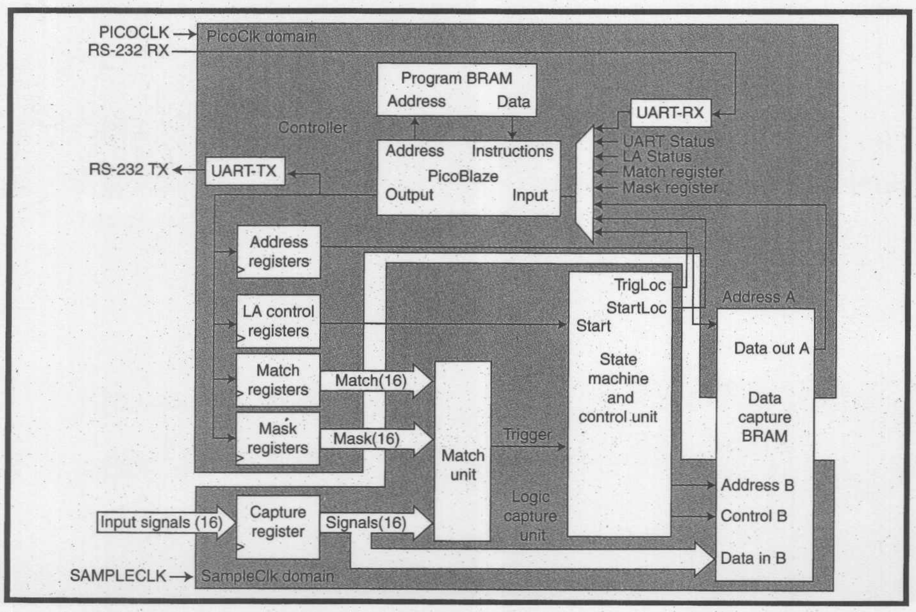

Figure 1—*There are only five signals in and out of the logic analyzer: TX, RX, PICOCLK, SAMPLECLK, and the 16 sample signals. The SampleClk and PicoClk domains are separated by the dual-ported nature of the data capture BRAM.*

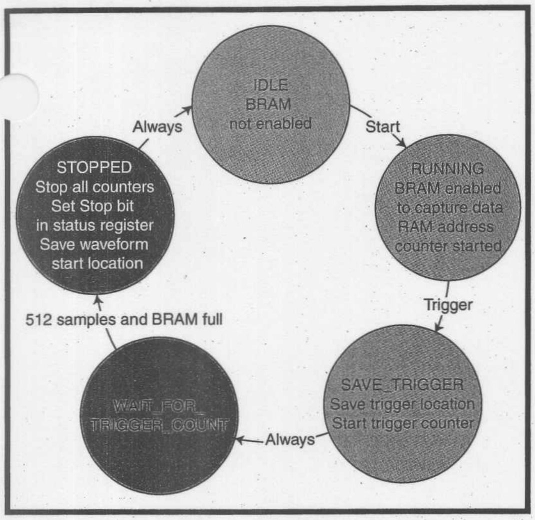

Figure 2—*The state machine controls the processes of writing captured data into the BRAM and capturing the trigger and waveform start locations.*

register and mask register. The match output is true if the input signal is equal to the match register bit and the mask register bit is 1. The match output is also true if the mask register bit is set to 0 (i.e., don't care). The match unit's output is true only if every single-bit compare module output is true.

The next part of the design consisted the state machine and control unit, which is responsible for starting and stopping the process of writing data to the block RAM (BRAM). The BRAM is a dual-ported embedded 18-Kb RAM. For the logic analyzer, I used port B in 1K × 16 bit mode to capture the data, with nothing connected to the data output pins. (As you'll see, the reading is achieved from the PicoBlaze through port A, which is organized as 2K × 8.)

The state machine is also responsible for saving the trigger location into a register as well as the BRAM address location where the displayed waveform starts (see Figure 2). Note that the conditions for exiting the WAIT_FOR_TRIGGER_COUNT state include having at least 512 samples after the trigger and having the BRAM full of data. So, if the trigger happens within the first 512 samples, the state machine will continue to fill the BRAM with data until it's full. This means that the trigger will not necessarily be in the middle of the captured waveform.

PicoBlaze CIRCUITRY

The 8-bit PicoBlaze microcontroller is extremely useful for complex state machines that aren't fast. I used it to communicate with the PC, to load the match and mask registers, to control the logic analyzer state machine, and to read the captured waveform from the BRAM. The PicoBlaze reads the BRAM from port A, which is set up as 8 bits wide. This simplifies the PicoBlaze's code.

The PicoBlaze program consists of a loop that waits for commands on the PC (see Table 1). The Dump command prints the trigger location register content, the start waveform register contents, a > symbol, the calculated trigger location, and the data, as shown in Listing 1.

Command	Description	Syntax
Match	Loads the Match register	MATCH hhhh
Mask	Loads the Mask register	MASK hhhh
Go	Starts the Logic Analyzer state machine	GO
Dump	Reads the content of the BRAM starting at the address located in the Start Waveform register. Sends it out the serial port.	DUMP

Table 1—*Use these commands to operate the logic analyzer via HyperTerminal. After loading the Match and Mask registers, type GO.*

Listing 1—*The output from the Dump command includes the > symbol (on the sixth line), which allows Excel to synchronize the data. The comments don't appear in the actual DUMP output.*

```
dump
0000Match     //Match register contents
00FFMask      //Mask register contents
0157Trigger   //Trigger location in BRAM
0002Start     //Waveform start location in BRAM
>             //Greater than symbol to allow Excel macro to
              //synchronize with the data to follow
0154          //Calculated trigger location
59AB          //First of 1,024 samples of data
59AB
59AB
59AB
59AC
59AC
59AC
59AC
59AD
59AD
59AD
```

EXCEL INTERFACE

I had thought about building an inexpensive logic analyzer for a long time. My first pass at designing one involved an old PLD from Intel. (Remember those?) Unfortunately, I didn't like the device's performance, so I shelved the project. When I was designing my SEB3, I figured it was time to resurrect the logic analyzer project. Technology had progressed a lot since my first attempt.

When I decided to build the logic analyzer with my new FPGA board I realized that the hardest part would be the user interface. Then I had an idea. Excel allows you to graph all sorts of things, so it must be able to graph some logic waveforms, I thought. Why not use Excel?

I'm not an Excel expert. I use it for budgeting, bills of materials, and design calculations. One thing that I've never been good at (never had to be) is graphing in Excel. So, if you're an Excel expert, please bear with me. It really does work out.

When I started to play with Excel, I discovered the XY scatter graph. This seemed ideal. All I needed was to get the data to enter Excel in the right format and then I could use the XY scatter graph. But getting the data in the appropriate format was a trade-off. Because I was using a serial port (38,400 bps) for communication from the PC to the FPGA board, I didn't want to send a complete comma-separated variable file with ones and zeros. Instead, I

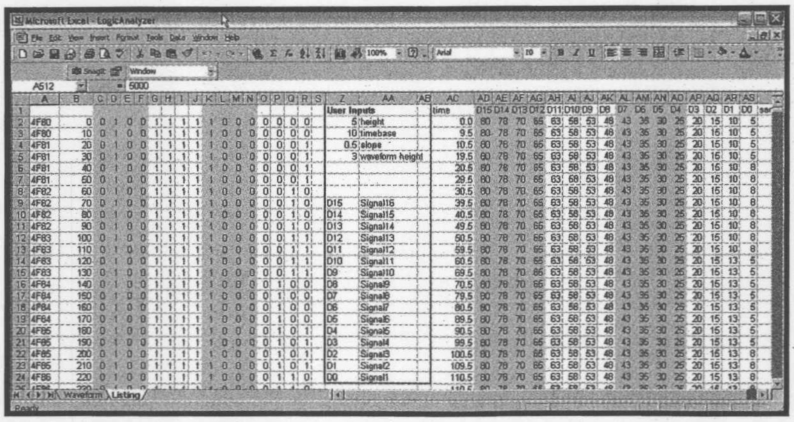

Photo 1—*The left side of the program shows the data input from the text file captured in HyperTerminal. The manipulated data is on the right. The values in the middle control the output waveform's display.*

opted to send a hex value for each sample.

Listing 1 shows the beginning of a waveform file from the FPGA board. It was captured in HyperTerminal and saved as a text file. Because the input was being received in hex format, I needed to create a macro that expanded these values into a binary representation spread out across 16 cells. That was relatively easy. Next, I tried graphing it. I must admit it didn't look pretty. Excel just shoved the lines over each another.

Next, I needed to space out the lines, which required translating the data so that they didn't overlap. I then added a timescale and graphed the results. The result was much better, but the waveforms were triangular. From one time step to another, you could go from zero to one to zero again and Excel would draw this as a sawtooth waveform. Not bad, but not great. So, to correct this I duplicated each sample of data twice. For example, at time sample 10 ns, there was now a time sample that represented a small time before 10 ns and another time sample that represented a small time after 10 ns. This small time before and after the actual sample is stored in a cell for easy access.

To read the captured text file and perform all the necessary data manipulations, I created a macro that's invoked by pressing CTRL+Q. This brings up an Open File dialog box where you must select the captured text file. After selecting the file, the file is read, the data is manipulated, and the waveform is drawn. The process takes approximately 5 s on a Centrino 1.6-GHz laptop.

Earlier, you may have thought that having only 1,024 samples is limiting (and you still may be right), but discerning anything when looking at the full 1,024 samples can be difficult. I needed the ability to zoom in and out as well as pan left and right in the sample space. I played with different ways of doing this, including regenerating the graph each time I recalculated the display window. But that also made it difficult for the macros to know the current graph so that manipulations could occur because each graph was given a new

name. An Excel expert would say the best way to do this would be to change the scale the x-axis. After I discovered this, it took just a couple of hours to code Zoom In (CTRL+I), Zoom Out (CTRL+O), Pan Left (CTRL+L), Pan Right (CTRL+R), and View All (CTRL+A). This worked out extremely well.

Photo 1 shows the listing sheet in the Excel file. Column A is the captured data in hex format. Columns B through R are the data broken down into ones and zeros. Columns Z and AA contain constants that you can change to affect the output waveform, as well as a place to name the signals so that the names are shown on the waveform display. Columns Z and AA also give cells that allow you to change the zoom factor and the pan step size. There's also a location that shows where the trigger should be displayed on the waveform (not shown in Photo 1). Columns AD to AS show the waveform values that have gone through the aforementioned manipulations. The end result is shown in Photo 2.

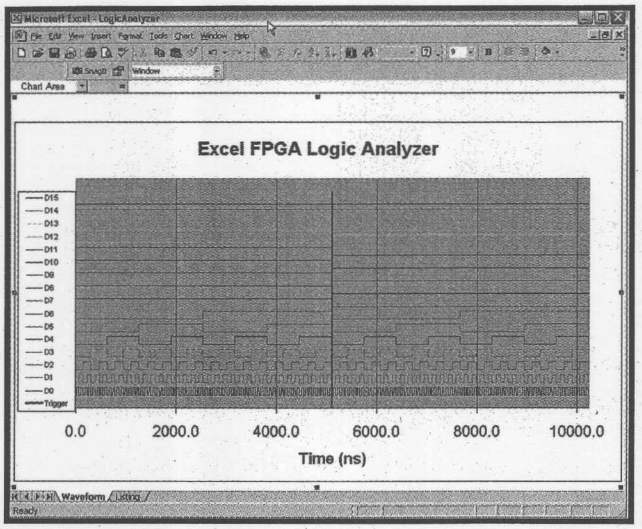

Photo 2—*As you study the captured waveform, note that the logic analyzer was set up to trigger on 5000h with a mask of FFFFh. The trigger location is clearly visible in red. This is all achieved with a standard Excel XY scatter graph.*

DESIGN TIME

Ready to start a project of your own? First, add the PicoBlaze logic analyzer module to your FPGA design and connect the signals you want to observe. Then connect the RX and TX serial signals to your serial port. Following this, connect the PicoBlaze's clock (50 MHz) and SampleClk to your sample clock. Next, you must resynthesize, place, and route your design.

The next step involves HyperTerminal. Set up the mask and match values and then type `GO`. After the traces are captured, use HyperTerminal to capture to a text file, type `DUMP`, and then stop the capture in HyperTerminal. Following this, click on the listing sheet in Excel (with the logic analyzer spreadsheet open) and press CTRL+Q. When the file dialog box opens, find and open the text file captured with HyperTerminal. This will read the data in and graph it on the waveform sheet. Click on the Waveform Sheet tab and your waveform will appear. Use the control keys to move around in the waveform.

I used the circuitry in Figure 3 to test the logic analyzer's circuitry. The PicoBlaze logic analyzer and clockgen modules are all you need. The clockgen block is the digital clock manager (DCM) used to generate the 50-MHz clock for the PicoBlaze, the 100-MHz sample clock, and a 25-MHz counter clock. The Count16 module is a 16-bit up counter. The Buf16a module simply buffers the clock outputs to some user I/O on the board.

ELA SOLUTION

Debugging a complex piece of electronics like the internals of an FPGA can be difficult. Not being able to see what's happening inside the FPGA makes the job next to impossible. But with an effective tool like the ELA, you can see what's happening in the FPGA and take measures to correct any problems.

I hope you find the ELA useful for your FPGA projects. You don't need an expensive external logic analyzer, multichannel digital storage oscilloscope, or high-cost embedded logic analyzer. Best of all, this version is free if you already have Excel! You may download the FPGA source files and the Excel spreadsheet from the *Circuit Cellar* web site. ■

Author's note: I plan to release a full-featured embedded logic analyzer through Dulse Electronics (www.dulseelectronics.com). Suggestions are welcome.

Philip Nowe earned a Bachelor's degree in engineering at Carleton University in Ottawa. He has been working in the hardware design industry more than 20 years. Philip runs Dulse Electronics, which provides affordable FPGA boards and tools. You may contact him at pnowe@dulseelectronics.com.

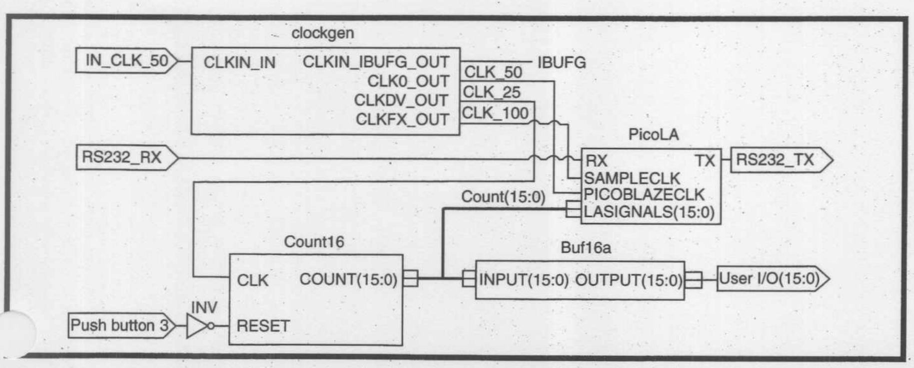

Figure 3—*The input is a 16-bit counter. For debugging purposes, Buf16a drives the counter output to the user I/O section of the FPGA board. The clockgen block is a digital clock analyzer with three separate clock outputs: 100 MHz for SampleClk, 50 MHz for PicoClk, and 25 MHz for the counter clock.*

PROJECT FILES

To download the code, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2005/179.

SOURCES

Spartan-3 Experimenter's Board
Dulse Electronics
www.dulseelectronics.com

PicoBlaze microcontroller, Spartan-3 FPGA, and WebPACK
Xilinx
www.xilinx.com